結び目の作図とその簡素化 (数式処理と数学研究への応用)

AUTHOR(S):

本間, 龍雄; 森川, 治; 落合, 豊行

CITATION:

本間, 龍雄 ...[et al]. 結び目の作図とその簡素化 (数式処理と数学研究への応用). 数理解析研究所講究録 1980, 406: 52-70

ISSUE DATE:

1980-12

URL:

http://hdl.handle.net/2433/102350

数理解析研究所講究録
406巻 1980年 52-70

結び目の作図とその簡素化

東工大・理　本間　龍雄
〃　森川　治
阪大・理　落合　豊行

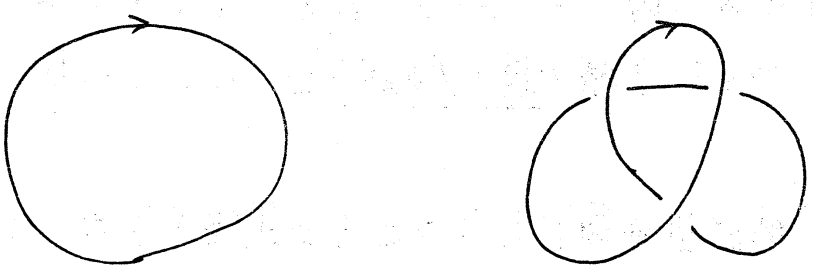
0. はじめに.

結び目理論において、結び目の判別の仕方はいろいろあり、又それぞれに不変量が提案されている。しかし、完全に結び目を他の結び目と区別できる不変量は、まだ見つかっていない。結び目のうち特に図1のようないわゆる“ほどけた結び目”すなわちtrivialな結び目と、図2のような“ほどけない結び目”との区別については、ある程度知られている。すなわち、ある与えられた結び目が、trivialであるか否かは、判定できるわけであるが、その判定のアルゴリズムは、あまり知られていない。これに対し、Haken [3], De Gruse [4] らは、normal surface という概念を導入し、trivialな結び目か否かを判定するアルゴリズムを提案した。しかし、この方法は完全ではあるが、実際の結び目を判定することは、天文学的な場合数に対しすべて調べる必要があり、全く

実用にならない。又、結び目群（knot group）という不変量は、次の定理が示す通り、この判定に関する完全な不変量となっているが、今度は、群が可換か非可換かを判定することが問題となり、やはりこの判定アルゴリズムも知られていないので、実用にならない。

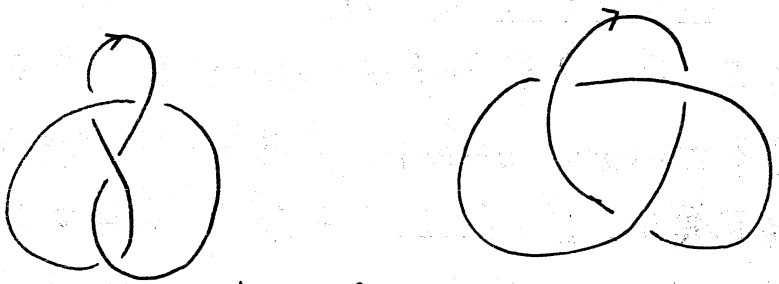
定理. 結び目Kが、trivial である必要十分条件は、結び目群πKがアーベル群であることである。

そこで、私達はもっと実用的なアルゴリズムを研究中である。ここで述べるのは、その中間結果である。



trivial な結び目

図 1.



trivialでない結び目

図 2.

1. 結び目の定義.

結び目とは、元来ひもを結び合わせた部分をさすものだが、ここでは、始点と終点を一致させたいわば"結び輪"を結び目という。数学的に定義すると、空間としてユークリッド空間$\mathbb{R}^3$または、$\mathbb{R}^3$を1点コンパクト化した3次元球面S^3内における、輪S^1と同相な図形を結び目ということになる。ここで、空間S^3, 輪S^1共に向きが付いているものとする、すなわち、

定義 S^1, S^3は共に向き付けられていて、gをS^1からS^3への埋め込み写像. すなわち、S^1と$g(S^1)$が同相となる写像のとき、$g(S^1)$を結び目(knot)といい、この$g(S^1)$をKと表わす。

次に、結び目が同じということを定義するが、これはわかりやすく言えば、結び目が連続的に変形して互いに移り合えることをいうものである.

定義 結び目K, K'が同値とは、$K=g(S^1)$, $K'=g'(S^1)$としたとき、gとg'がambient isotopicであることである。

gとg'がambient isotopicであるとは、次の3条件をみたす連続写像$F: S^3\times[0,1]\longrightarrow S^3$が存在することである.

1. 任意の$t\in[0,1]$に対し、$F(x,t)$を$f_t(x)$と書くことにすると、$f_t: S^3\longrightarrow S^3$がhomeomorphismとなっている。

2. f_0 は恒等写像

3. f_1 と g との合成写像 $f_1 \circ g : S^1 \hookrightarrow S^3$ が g' に等しい。

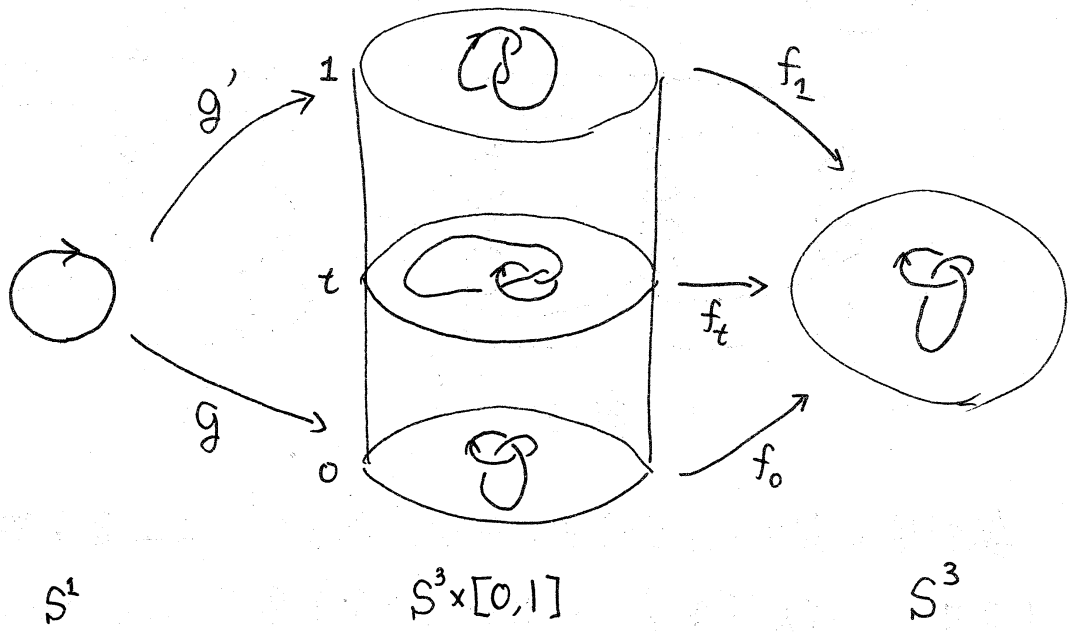


図3.

定義は以上ですが、ここではKは有限折れ線で表わせるものだけを対象とし、図4のような物は対象としない。

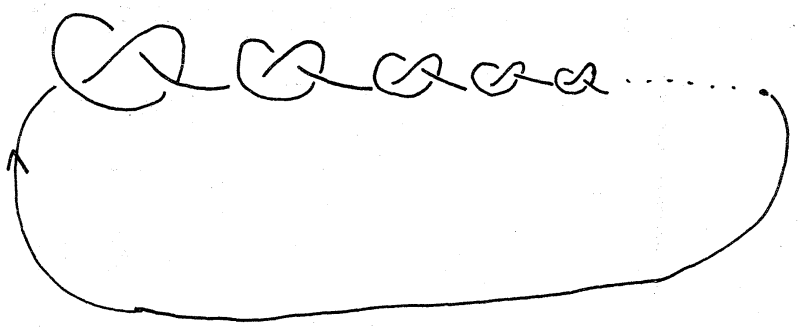


図4.

例. 次の結び目は全て異なる.

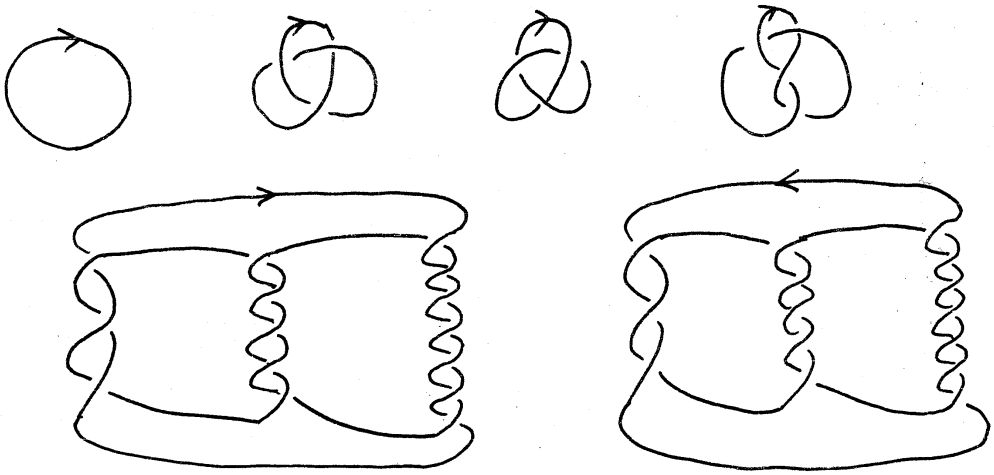


図 5

結び目は $\mathbb{R}^3$ (又は S^3) 内のものであるが. これを平面 $\mathbb{R}^2$ (又は. 2次元球面 S^2) へ射影した射影図 (knot form) により話をすすめる. このとき. 本当に射影してしまっては、交差点の上下関係がわからなくなってしまうので、上にある方の部分を上道 (overpass) といい、下の方を下道 (underpass) といって区別することにする。図6の例では、

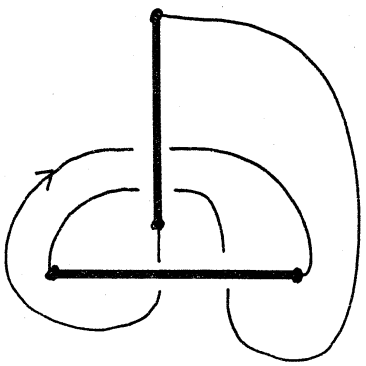


図 6

2本ずつの overpass, underpass により. 結び目が与えられているので. この射影図を. 2-bridge form という。一般に、n本ずつの場合. n-bridge form という. 以下. K は射影図を表わすと

し、特にことわりのない限り、結び目と射影図を同一視して述べる.

結び目の同値性に対する次の定理がある.

定理　結び目K, K'が同値である

⟺　Kは次の3つの操作を有限回行なってK'に変形できる。

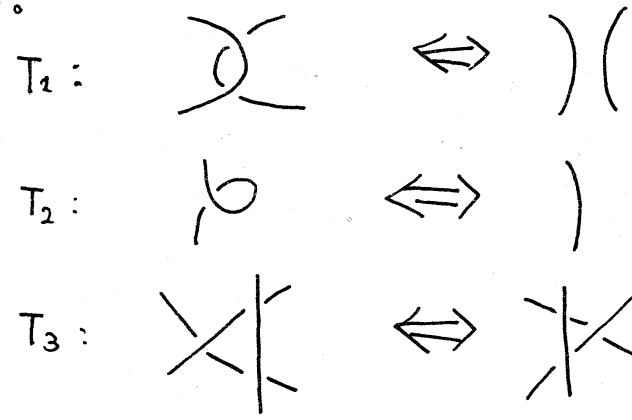
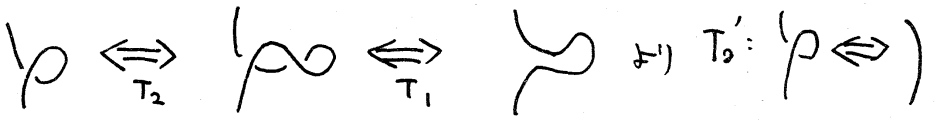


図7

(注). 

も含まれる。

2.　同値性の問題

問題1.　結び目K, K'が与えられたとき. これらが同値であるか否かを判定せよ。

問題1に対し.“KとK'は同値である”と断定できるのは、実際に定理のT_1, T_2, T_3 の組合せで、KからK'への変形をつくってやる以外の判定法は、ほとんど知られていない.

しかし逆に、同値でないとの断定には、種々の不変量が提案されていて、例えば図2の2つの結び目は、Alexander Polynomial という不変量がそれぞれ、t^2-3t+1, t^2-t+1 と異なるため結び目も同値でないとわかる。

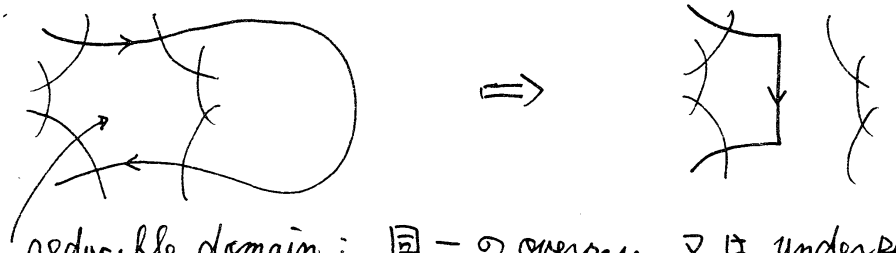
問題1は大変難かしいので、部分的問題として、次のものがある。

問題2. 結び目Kが与えられたとき、Kがtrivialな結び目であるか判定せよ。

これに対しては、はじめにも述べたが、Haken, DeGruse により解は存在するが、実用的なものは見つかっていない。部分的解として次のものがある。

定理 (本間, 落合, 高橋)[6]

結び目Kが、3-bridge form によって与えられているとき、Kがtrivialな結び目と同値ならば、reducible domain によるreduction により、Kをtrivialな結び目 ◯ に変形できる。



reducible domain: 同一のoverpass又はunderpassに属する2辺以上をもつ面. 図8.

この reduction により、knot form の交差点数は減少するので、高々、交差点の数だけ実行すれば判定ができる。

この定理の操作は、3-bridge form でなくても可能であり、拡張が考えられるが、4-bridge form ですでに、その反例が、O. Q. Bupo (1977) [7] によって示されている。

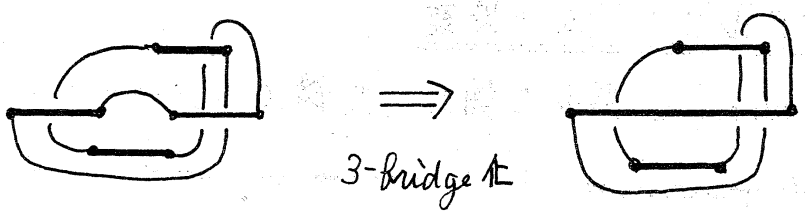


図9

しかし、この反例は容易に3-bridge化が行なえ、trivial な結び目であることが示せる。すなわち、定理の操作に、下記のような、bridge number 減少の操作を加えれば、この例は拡張性の否定とはならない。

・bridge number 減少の操作：　交互点をもたない overpass 又は underpass は前後の underpass 又は overpass の一部とみなし、bridge number を減少させる。

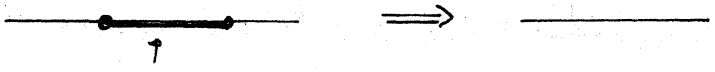


図10.

しかし、この操作を加えても、4-bridge form で反例が得られた。(森川. 1980 [8])

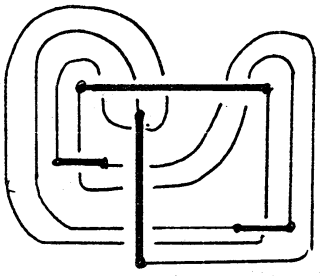
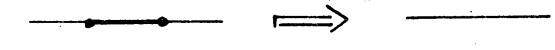


図11　4-bridge form での反例.

3. 新アルゴリズムの提案

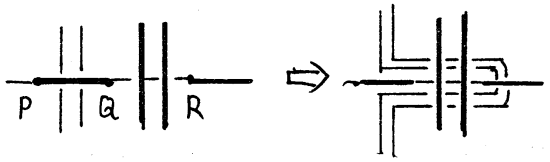
bridge number 減少を軸とした操作を次のように定める.

1. Short operation :



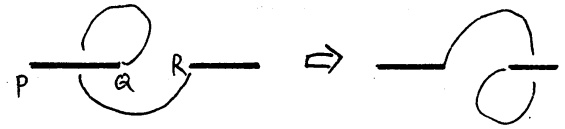
交差点のない overpass, underpass を前後の pass を短絡させることにより消去する.

2. move operation :



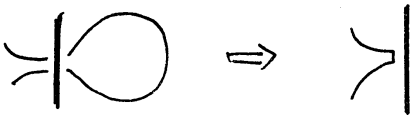
PQ間にある交差点をとなりの overpass のところまで動かしてやり. PQ を Short operation により消す.

3. change operation :

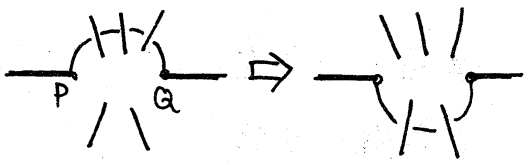


underpass QR を変形し、$PQ \cap QR = \{Q\}$ とする. これにより次に move operation が行なえ. bridge number が減少する。

4. annihilation :



overpass のとなり合った交差点が、同一の underpass と交差しているとき、underpass を変形して交差点を減らす。

5. color operation : 

underpass PQ を現在の状態より交差数の少ない他の underpass に変更する。(この underpass を見つけるとき、面に色をつけたことより、この名前となった。)

以上5つの操作によって結び目を変形することを、新アルゴリズムとした。これに対し、次の3つの目標を考えた。

1. 結び目Kが与えられたとき、このアルゴリズムに従ってKを変形してゆけば、trivial かるか判定できる。

2. さらに、Kが non-trivial のときも、bridge index という不変量が決定できる。

3. 2つの結び目の同値性が示せる。

以上3つの目標に対し、人力で数ケのサンプルで実験したところ、うまくゆくので、コンピュータ実験を行なうことになった。

4. コンピュータのために… 結び目のデータ構造

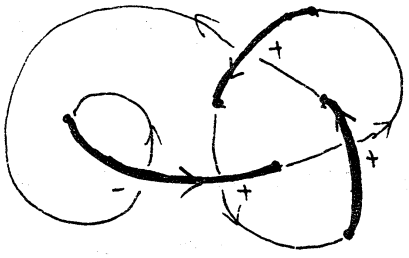
結び目Kは、次の情報により表現できる。

1. 交差点及び、overpass と underpass の境目。

2. これらのつながり具合、交わり具合。

3. 結び目の向き。

この情報を、細胞 cell = (OVnext, UDnext, orient) の3つ組によって記憶させる。



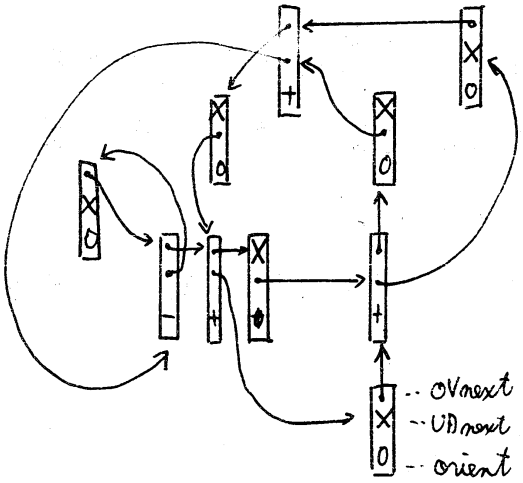


図12

OVnext は overpass でつながっている点を指すポインタ型変数, UDnext は underpass での点へのポインタ, orient は、交わり方で、  は+, は−, または は0とした。

結び目は全てこのようなデータによって表わせるが、任意のデータが結び目を表現しているとは限らない。そこで、次の定理がある。

<u>定理</u>　内部データが結び目を表現する必要十分条件は、このデータより導かれる oriented closed surface の genus が0であることである。

oriented closed surface は、oriented faces により構成される。oriented face の構成法は、紙面の都合上、省略する。

5. 使用言語

本実験は、操作の定義から明らかなように、各操作にuniqueⅠ性はなく、1つの状況から、多数の操作後の状況が生ずる。この各々をチェックする必要があるので、次のような流れ図となる。

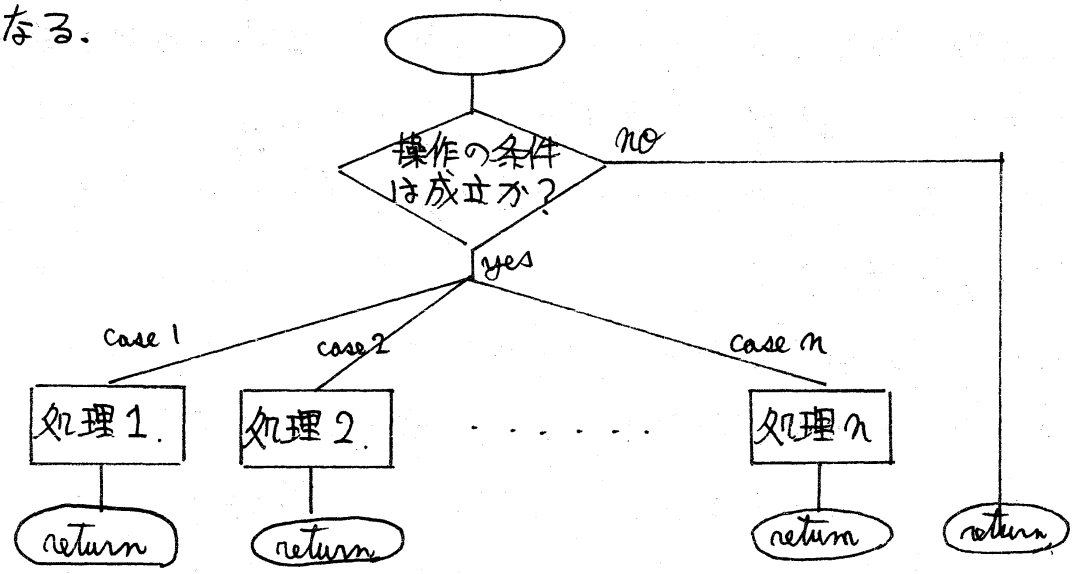


図13

このルーチンの入口では1台だった計算機が、出口ではn台の計算機に分裂して、各々別々に後の処理を行なってゆく必要がある。そこで、これを可能にする"CPU分裂命令"をもった言語を製作した。従来は、複数個の独立した仕事をさせるときでも、それらの処理順序を指定して、ともかく1列に並べなければならなかったが、この言語ではシステムが自動的に処理順序を定めて行なってくれるので、ユーザは、概念上複数台のコンピュータが同時に処理を行なっていると

思ってプログラミングができる。このため、プログラミングが大変楽になった。

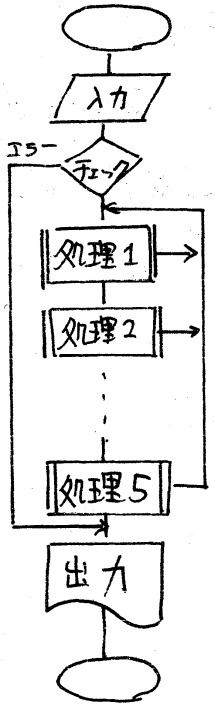
○CPU分裂命令の実際.

計算機は実際は1台なので、分裂命令があっても、case1の処理を続行する。しかし、1台目の計算機の処理が終了した時点で、Job終了とせずに、先ほどのcase2の処理をつづけて行なう。この2台目の処理も終ると3台目, 4台目と順次以下同様に行ない、すべてのコンピュータが終ったとき、初めてJob終了となる。ユーザにとっては、1台目, 2台目がどういう順序で処理されようと関係なく、結果としてすべてをつくしてくれればよく、この方法で問題ない。さて、ここで2台目のコンピュータが動き出すためには、何が必要かというと、1台目のコンピュータが分裂命令を出した直前の状態を再現してやればよいことになる。この"状態"は、プログラムによって、何が必要で何が不必要な情報かが異なるので、システムで行なうときは、全てを記憶する必要があり、メモリ効率がよくない。そこで、本言語では、必要なデータをユーザが指定し、ユーザプログラムで記憶させるという方式をとった。このため、コンパクトにして小さく記憶させることができ、メモリの有効利用が計られた。私のプログラムの場合、実際記憶したのは、結び目の内部表現, 作業ポインタ,

サブルーチンのネストの状況，2台目のコンピュータが走り出すプログラムアドレス，さらに、デバッグのためのメッセージである。そこで、それらを記憶するルーチンPush，逆に記憶をもとにもどすルーチンPopをユーザプログラム内に作成した。

○その他の命令.

算術命令は加減に限り、数は16ビット精度のintegerとした。用途がはっきりしているため、これで十分とした。変数はポインタ型と数値型の2種類．配列は1次元だけとした。文構造として、if-then-else文，for文，while文，repeat-until文を用いることにより、goto-lessプログラムとなるようにした。しかし、図14のように本実験では、何らかの処理（結び目の変形）を行なうと、再びループの初めからチェックするアルゴリズムのため、ループから飛び出すbreak文，ループの次回へスキップするnext文を採用した。この2文がなくても書くことはできるが．プログラムが煩雑になる恐れがあるので採用した。



ブロック構造は、element内と外の

2種類だけである。しかし、1つのelement内に複数のルーチンを入れることができるようにしたため、関連の深いルーチンは、同一のelement内に作成し、極力変数はlocal variableとして、外部からはさわれないようにし、モジュール化が計れた。

ポインタ型変数を採用したことにより、次の2種の記号!, @を次の意味として用いた。!名前は、名前のアドレス定数の意味で、A:=!B により、ポインタAがBを指すことになり、@名前は、名前の指しているアドレスの内容の意味で、C:=@A を実行すると、Cにポインタ Aの指している変数、ここではBの内容が代入される。すなわち、! は代入文の右辺にあり、左辺にはポインタ型変数が来る。また、@の直後の名前は、ポインタ型変数である必要がある。

システムで用意したサービス・ルーチンは、standard I/Oモジュールで、getC, getNo, PutC, putNo, putNoF, putStr, putTab, putback の8つである。Cはcharacter, Noはnumber, strはstring, Tabはtabration, NoFはnumber with formatの略である。文法については表の通りである。

なお、本言語作成に当っては、本学科（情報科学学科）・木村研の学生、久野 靖 氏の協力を得た。

文法：

エレメント ＝ element 名前 ; [相制 ;] ○○○ 文 {;} ○○○ end.

相制 ＝ (global | external) 名前 {,} ○○○

文 ＝ procedure 名前 ; 文 {;} ○○○ end |
if 条件 then 文 [else 文] |
for 文 [step 文] [while 条件] do 文 [until 条件] |
while 条件 do 文 [until 条件] |
repeat 文 [until 条件] |
break | next | 式 |
call [@] 名前 [(式)] | return [(式)] |
(文 {;} ○○○) | < コード文 {;} ○○○ >

コード文 ＝ アセンブラ

条件 ＝ (plus | notplus | minus | not minus | zero | notzero) [(式)] |
式 (> | ¬> | < | ¬< | = | ¬=) 変数

式 ＝ [(レジスタ | 変数) :=] ○○○ ([+|-] 数 | レジスタ
| 変数) [(+ | -) (数 | レジスタ | 変数)] ○○○

レジスタ ＝ R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7

変数 ＝ [! | @] 名前 [(式)]

6. 実験結果

Alexander-Briggs の knot table の結び目をチェックしたところ、8_4 までは全て同一の bridge number，交点数の reduced knot form が得られたが、8_5 は異なる交点数のものが見つかった。すなわち、目標3の反例が見つかったわけだが、さらに 8_{21} まで行なったところ、全て bridge number は同一であったので、目標1，2については否定されていない。

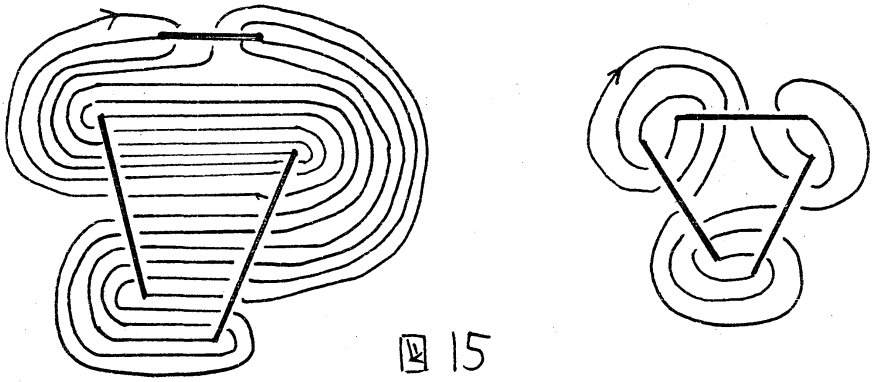


図15

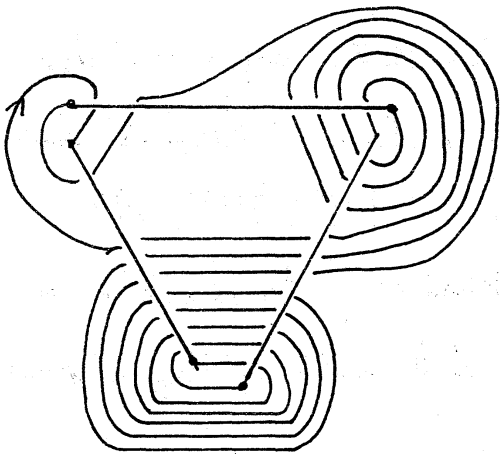
この実験のアルゴリズムより明らかなように、次の結果も得られた。

定理　任意の non-trivial な結び目 K は次のような結び目群 πK の表示が可能である：$\pi K = \langle x_1, x_2, \cdots, x_n \mid r_1, \cdots, r_{n-1} \rangle$ において、$\theta : \pi K \longrightarrow \pi K/[\pi K, \pi K] = \mathbb{Z}$: homomorphism とすると、任意の i, j に対し、$\theta(x_i) = \theta(x_j)$．$r_i$ の word は，同一のアルファベットは並んで出現しない、又，奇数回現れるアルファベットは，必ず3回以上現われている。

この定理の条件をみたす表示は、reduced knot form から容易に求められる。この結果を利用して、結び目が、trivial でないことを示すアプローチとして、次の予想がある。

予想: 結び目Kを trivial でないとし、$\pi K = \langle x_1, \cdots, x_n \mid r_1, \cdots r_{n-1} \rangle$ を定理の条件をみたす群表示とする。このとき、$G = \langle x_1, \cdots, x_n \mid r_1, \cdots, r_n, x_i^2 = 1 \ (i=1,2,\cdots,n) \rangle$ とおくと、Gも非可換群であろう。

この予想は、次の例では正しい。



$$\pi K = \langle x, y, z \mid xyx^{-1}(zy^{-1})^2 z^{-1}(yz^{-1})^2 xy^{-1},\ z(x^{-1}z)^3 y^{-1}xy^{-1}x^{-1}y(z^{-1}x)^3 \rangle$$

$$G = \langle x, y, z \mid (xy)^3,\ (yz)^5, (zx)^7 \rangle$$

図 16

Gが非可換なことは容易に示せる。よってこの結び目は、trivial でない。しかし、Alexander polynomial という不変量では、この結び目も、trivial な結び目も共に1であり区別できない。現在、この例の結び目を拡張した、ある種

の結び目の集合（無限個）において予想が正しいことが示され、又、それらは全て、Alexander polynomial が1であるが、予想により non-trivial な結び目であることが示せることがわかっている。

参考文献

[1] Rolfsen: Knot and Links, Publish or Perish, Inc. 1976

[2] R.H. Crowell and R.H. Fox (H. Terasaka, H. Noguchi): 結び目理論入門, 岩波書店, 現代科学選書

[3] W. Haken: Theorie der Normalflecken, Acta math., 105 (1961) 245-375

[4] G. DeGruse: _____, Dep. Math. Colorad State Univ. Sep (1975)

[5] T. Homma, M. Ochiai: on relations of Heegaard diagrams and Knots, Math. Seminar Notes 6 (1978) 383-393

[6] T. Homma, M. Ochiai, M. Takahashi: An algorithm for recognizing S^3 in 3-manifold with Heegaard splitting of genus two, to appear

[7] О.Я. Виро: Гипотеза Володина-Кузнецова-Фоменко о диаграммах узлов трехмерной сферы не верна, Uspeki Mat. Nauk 32 (1977), No. 5 (197), 175-176

[8] O. Morikawa: A counterexample to a conjecture of Whitehead, to appear